EDA_with_Personal_Email_Analysis

November 27, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
!pip install mailbox
```

```
Collecting mailbox
  Downloading mailbox-0.4.tar.gz (4.1 kB)
  Preparing metadata (setup.py) … done
Building wheels for collected packages: mailbox
  Building wheel for mailbox (setup.py) … done
  Created wheel for mailbox: filename=mailbox-0.4-py3-none-any.whl
size=4684
sha256=8fac459ede30179c2914f0609ad51234c7551ca8505d9900bb8fadc42302e1f9
  Stored in directory: /home/test/.cache/pip/wheels/06/cd/9a/64b75da2511d797260d
3b3cb8cfbf66e700119cc045a9be2c9
Successfully built mailbox
Installing collected packages: mailbox
Successfully installed mailbox-0.4
```

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

[ ]:
```
import mailbox
mboxfile = "gdrive/My Drive/Colab Notebooks/gmail.mbox"

mbox = mailbox.mbox(mboxfile)
mbox
```

[ ]:
```
for key in mbox[0].keys():
  print(key)
```

[ ]:
```
import csv

with open('mailbox.csv', 'w') as outputfile:
  writer = csv.writer(outputfile)
  writer.writerow(['subject','from','date','to','label','thread'])
  for message in mbox:
    writer.writerow([message['subject'], message['from'],  message['date'],␣
 ↪message['to'],  message['X-Gmail-Labels'], message['X-GM-THRID']])
```

[ ]:
```
dfs = pd.read_csv('mailbox.csv', names=['subject', 'from', 'date', 'to',␣
 ↪'label', 'thread'])
```

[ ]:
```
dfs.dtypes
```

[ ]:
```
dfs['date'] = dfs['date'].apply(lambda x: pd.to_datetime(x, errors='coerce',␣
 ↪utc=True))
```

[ ]:
```
dfs = dfs[dfs['date'].notna()]
```

[ ]:
```
dfs.to_csv('gmail.csv')
```

[ ]:
```
dfs.info()
```

[ ]:
```
dfs.head(10)
```

[ ]:
```
dfs.columns
```

[ ]:
```
import re

def extract_email_ID(string):
  email = re.findall(r'<(.+?)>', string)
  if not email:
    email = list(filter(lambda y: '@' in y, string.split()))
  return email[0] if email else np.nan
```

[ ]:
```
dfs['from'] = dfs['from'].apply(lambda x: extract_email_ID(x))
```

[ ]:
```
myemail = 'itsmeskm99@gmail.com'
dfs['label'] = dfs['from'].apply(lambda x: 'sent' if x==myemail else 'inbox')
```

[ ]:
```
dfs.drop(columns='to', inplace=True)
```

[ ]:
```
dfs.head(10)
```

[ ]:
```
import datetime
import pytz

def refactor_timezone(x):
  est = pytz.timezone('US/Eastern')
  return x.astimezone(est)
```

[ ]:
```
dfs['date'] = dfs['date'].apply(lambda x: refactor_timezone(x))
```

[ ]:
```
dfs['dayofweek'] = dfs['date'].apply(lambda x: x.weekday_name)
dfs['dayofweek'] = pd.Categorical(dfs['dayofweek'], categories=[
    'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
    'Saturday', 'Sunday'], ordered=True)
```

[ ]:
```
dfs['timeofday'] = dfs['date'].apply(lambda x: x.hour + x.minute/60 + x.second/
 ↪3600)
```

[ ]:
```
dfs['hour'] = dfs['date'].apply(lambda x: x.hour)
```

[ ]:
```
dfs['year_int'] = dfs['date'].apply(lambda x: x.year)
```

[ ]:
```
dfs['year'] = dfs['date'].apply(lambda x: x.year + x.dayofyear/365.25)
```

[ ]:
```
dfs.index = dfs['date']
del dfs['date']
```

[ ]:
```
print(dfs.index.min().strftime('%a, %d %b %Y %I:%M %p'))
print(dfs.index.max().strftime('%a, %d %b %Y %I:%M %p'))

print(dfs['label'].value_counts())
```

[ ]:
```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
```

[ ]:
```
def plot_todo_vs_year(df, ax, color='C0', s=0.5, title=''):
  ind = np.zeros(len(df), dtype='bool')
  est = pytz.timezone('US/Eastern')

  df[~ind].plot.scatter('year', 'timeofday', s=s, alpha=0.6, ax=ax, color=color)
  ax.set_ylim(0, 24)
  ax.yaxis.set_major_locator(MaxNLocator(8))
  ax.set_yticklabels([datetime.datetime.strptime(str(int(np.mod(ts, 24))),␣
 ↪"%H").strftime("%I %p") for ts in ax.get_yticks()]);

  ax.set_xlabel('')
  ax.set_ylabel('')
  ax.set_title(title)
  ax.grid(ls=':', color='k')

  return ax
```

[ ]:
```
sent = dfs[dfs['label']=='sent']
received = dfs[dfs['label']=='inbox']
```

[ ]:

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(15, 4))

plot_todo_vs_year(sent, ax[0], title='Sent')
plot_todo_vs_year(received, ax[1], title='Received')
```

[ ]:

```
def plot_number_perday_per_year(df, ax, label=None, dt=0.3, **plot_kwargs):
    year = df[df['year'].notna()]['year'].values
    T = year.max() - year.min()
    bins = int(T / dt)
    weights = 1 / (np.ones_like(year) * dt * 365.25)
    ax.hist(year, bins=bins, weights=weights, label=label, **plot_kwargs);
    ax.grid(ls=':', color='k')
```

[ ]:

```
from scipy import ndimage

def plot_number_perdhour_per_year(df, ax, label=None, dt=1, smooth=False,
                      weight_fun=None, **plot_kwargs):

    tod = df[df['timeofday'].notna()]['timeofday'].values
    year = df[df['year'].notna()]['year'].values
    Ty = year.max() - year.min()
    T = tod.max() - tod.min()
    bins = int(T / dt)
    if weight_fun is None:
        weights = 1 / (np.ones_like(tod) * Ty * 365.25 / dt)
    else:
        weights = weight_fun(df)
    if smooth:
        hst, xedges = np.histogram(tod, bins=bins, weights=weights);
        x = np.delete(xedges, -1) + 0.5*(xedges[1] - xedges[0])
        hst = ndimage.gaussian_filter(hst, sigma=0.75)
        f = interp1d(x, hst, kind='cubic')
        x = np.linspace(x.min(), x.max(), 10000)
        hst = f(x)
        ax.plot(x, hst, label=label, **plot_kwargs)
    else:
        ax.hist(tod, bins=bins, weights=weights, label=label, **plot_kwargs);


    ax.grid(ls=':', color='k')
    orientation = plot_kwargs.get('orientation')
    if orientation is None or orientation == 'vertical':
        ax.set_xlim(0, 24)
        ax.xaxis.set_major_locator(MaxNLocator(8))
        ax.set_xticklabels([datetime.datetime.strptime(str(int(np.mod(ts,␣
 ↪24))), "%H").strftime("%I %p")
                            for ts in ax.get_xticks()]);
    elif orientation == 'horizontal':
        ax.set_ylim(0, 24)
        ax.yaxis.set_major_locator(MaxNLocator(8))
        ax.set_yticklabels([datetime.datetime.strptime(str(int(np.mod(ts,␣
 ↪24))), "%H").strftime("%I %p")
                            for ts in ax.get_yticks()]);
```

[ ]:

```
class TriplePlot:
  def __init__(self):
    gs = gridspec.GridSpec(6, 6)
    self.ax1 = plt.subplot(gs[2:6, :4])
    self.ax2 = plt.subplot(gs[2:6, 4:6], sharey=self.ax1)
    plt.setp(self.ax2.get_yticklabels(), visible=False);
    self.ax3 = plt.subplot(gs[:2, :4])
    plt.setp(self.ax3.get_xticklabels(), visible=False);

  def plot(self, df, color='darkblue', alpha=0.8, markersize=0.5, yr_bin=0.1,␣
 ↪hr_bin=0.5):
    plot_todo_vs_year(df, self.ax1, color=color, s=markersize)
    plot_number_perdhour_per_year(df, self.ax2, dt=hr_bin, color=color,␣
 ↪alpha=alpha, orientation='horizontal')
    self.ax2.set_xlabel('Average emails per hour')
    plot_number_perday_per_year(df, self.ax3, dt=yr_bin, color=color,␣
 ↪alpha=alpha)
    self.ax3.set_ylabel('Average emails per day')
```

[ ]:

```
import matplotlib.gridspec as gridspec
import matplotlib.patches as mpatches

plt.figure(figsize=(12,12));
tpl = TriplePlot()

tpl.plot(received, color='C0', alpha=0.5)
tpl.plot(sent, color='C1', alpha=0.5)
p1 = mpatches.Patch(color='C0', label='Incoming', alpha=0.5)
p2 = mpatches.Patch(color='C1', label='Outgoing', alpha=0.5)
plt.legend(handles=[p1, p2], bbox_to_anchor=[1.45, 0.7], fontsize=14,␣
 ↪shadow=True);
```

[ ]:

```
counts = dfs.dayofweek.value_counts(sort=False)
counts.plot(kind='bar')
```

[ ]:

```
addrs = received['from'].value_counts()

addrs[0:4]
```

[ ]:

```
plt.figure(figsize=(12,12));

tpl = TriplePlot()

labels = []
colors = ['C{}'.format(ii) for ii in range(9)]
idx = np.array([1,2,3,7])
for ct, addr in enumerate(addrs.index[idx]):
    tpl.plot(dfs[dfs['from'] == addr], color=colors[ct], alpha=0.3, yr_bin=0.5,␣
  ↪markersize=1.0)
    labels.append(mpatches.Patch(color=colors[ct], label=addrs[0:4], alpha=0.5))
plt.legend(handles=labels, bbox_to_anchor=[1.4, 0.9], fontsize=12, shadow=True);
```

[ ]:

```
sdw = sent.groupby('dayofweek').size() / len(sent)
rdw = received.groupby('dayofweek').size() / len(received)

df_tmp = pd.DataFrame(data={'Outgoing Email': sdw, 'Incoming Email':rdw})
df_tmp.plot(kind='bar', rot=45, figsize=(8,5), alpha=0.5)
plt.xlabel('');
plt.ylabel('Fraction of weekly emails');
plt.grid(ls=':', color='k', alpha=0.5)
```

[ ]:

```
import scipy.ndimage
from scipy.interpolate import interp1d

plt.figure(figsize=(8,5))
ax = plt.subplot(111)
for ct, dow in enumerate(dfs.dayofweek.cat.categories):
    df_r = received[received['dayofweek']==dow]
    weights = np.ones(len(df_r)) / len(received)
    wfun = lambda x: weights
    plot_number_perdhour_per_year(df_r, ax, dt=1, smooth=True, color=f'C{ct}',
                                  alpha=0.8, lw=3, label=dow, weight_fun=wfun)

    df_s = sent[sent['dayofweek']==dow]
    weights = np.ones(len(df_s)) / len(sent)
    wfun = lambda x: weights
    plot_number_perdhour_per_year(df_s, ax, dt=1, smooth=True, color=f'C{ct}',
                                  alpha=0.8, lw=2, label=dow, ls='--', weight_fun=wfun)
ax.set_ylabel('Fraction of weekly emails per hour')
plt.legend(loc='upper left')
```

[ ]:

```
from wordcloud import WordCloud

df_no_arxiv = dfs[dfs['from'] != 'no-reply@arXiv.org']
text = ' '.join(map(str, sent['subject'].values))
```

[ ]:

```
stopwords = ['Re', 'Fwd', '3A_']
wrd = WordCloud(width=700, height=480, margin=0, collocations=False)
for sw in stopwords:
    wrd.stopwords.add(sw)
wordcloud = wrd.generate(text)

plt.figure(figsize=(25,15))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.margins(x=0, y=0)
```